Large File Distributor

Sorakris Chaladlamsakul & Payut Tanyaluck

December 13, 2016

1 INTRODUCTION

Distributing a large file (10+ GB) to tens or even hundreds of machines interconnected on the same local-area network (LAN) can be very challenging. It is dreadful to designate a single machine as a master while letting tens or even hundreds of machines asking for the gigantic file. How would you do it? An easy idea is that we have to somewhat try to host the file to other machines and make them be the master as well or Peer-to-peer (P2P). Peer-to-peer (P2P) networking is one of the solution and used in this project. Since Peer-to-peer (P2P) computing or networking is a distributed application architecture that partitions tasks or workloads between peers.

2 GETTING STARTED

(1) Discovery: How can participating machines discover some special machine or the other machines that are their peers?

To discover other machine in the same local-area network the program operate this function by using UDP broadcasting.

(2) Transmission

Java implementation of the BitTorrent protocol, Ttorrent to transmit file.

A UDP file transfer is used to send small .torrent file

(3) Progress Tracking

Create a multithreaded Server on master and every machine keep telling their loading progress to that master (“thread per connection”). Less time is spent outside the accept() call. Long running client requests do not block the whole server

TO RUN: 1. run the .jar file on every machine eg. java -jar …. .jar

2. Assign a machine by opening another terminal and run the jar file follow with the file you wanted to share

Eg. java -jar …. .jar xyz.jpeg

3 IMPLEMENTATION AND DESIGN

(1) *Discovery*: To discover other machine in the same local-area network the program operate this function by using UDP broadcasting. Every machine basically keep shouting and every machine keep listening. However, everyone is set to shout every 1.5 seconds (using Thread.sleep()) and always keep their ears open. Furthermore, every machine is broadcasting through every network interfaces.

*Server implementation: *DiscoveryThread.java*

1. Open a socket on the server that listens to the UDP requests. (We've chosen 6789)
2. Make a loop that handles the UDP requests and responses
3. Inside the loop, check the received UDP packet to see if it's valid
4. Still inside the loop, send a response to the IP and Port of the received packet

*Client implementation: *UDPTalker.java*

1. Open a socket on a random port.
2. Try to broadcast to the default broadcast address (255.255.255.255)
3. Loop over all the computer's network interfaces and get their broadcast addresses
4. Send the UDP packet inside the loop to the interface's broadcast address
5. Wait for a reply
6. When we have a reply, check to see if the package is valid
7. When it's valid, get the package's sender IP address; this is the server's IP address
8. CLOSE the socket! We don't want to leave open random ports on someone else's computer

(2) *Transmission*: To transmit, master machine is ask which networkinterface it want to transfer the file over. Then the master machine have to create .ttorrent file and tell other (using UDP broadcasting)to come and download it via UDP server and client **SimpleFileClient.java SimpleFileServer.java*. And the real transmission process will be using the ttorrent to distribute the real file **ttorrentClient.java ttorrentServer.java*.

MASTER MACHINE

1. Ask for the interface that want to transfer the file to.
2. The master machine will create tracker and .torrent file **ttorrentTracker.java*
    a. First, Create a .torrent file with:
       File source, int pieceLength, List<List<URI>> announceList, String createdBy
    b. Then, instantiate a Tracker object with the por 6969 for it to listen on.
    c. Finally, for each torrent you wish to announce on this tracker, simply created a TrackedTorrent object and pass it to the tracker.announce() method:
    d. After it is done it will trigger step 3 to start.
3. The master machine then also seed for other to be able to download. *ttorrentClient.java
    a. This will be run through threadpool but will run when step 2 is done.
4. Create a UDP server local server hosting the .torrent file *SimpleFileServer.java
    a. Will be run in threadpool but will run when step 3 is done.
5. Tell other that tracker is created and .torrent file is ready to be download.**UDPTrigger.java*
    a. Work like UDPTalker but instead of broadcasting "DISCOVER_SERVER_REQUEST" it send "DISCOVER_SERVER_TRIGGER"
    b. This will run in threading
    c. Will change the status of *trackerStatus* to be true and will trigger the master to run seed via *ttorrentClient.java* will be multithreading.

CLIENT MACHINE

6. Then there will be machines that are listening **DiscoveryThread.java*
    a. Every machine that is listening via the discovery part will start asking and loading the .ttorrent using the IP provided from the datagram (packet.getAddress().getHostAddress()).**SimpleFileClient.java*
7. Client will starting downloading and sharing *ttorrentClient.java
    a. First, instantiate the Client object
    b. Set 0.0 for download and uploadspeed limits so there is no limit
    c. call client.share() for the client to be loading and sharing at the same time
    d. this.client.waitForCompletion();

(3) *Progress Tracking*:

MASTER MACHINE

1. After step 5 above master machine will create a multithreading server on the master machine in order to receive progress of each client.*ProgTrackMultiThrdServer.java #At first we use single thread server and it tear down
    a. This code uses a "thread per connection" design

CLIENT MACHINE

1. Inside step 7 while the client is downloading and sharing it will output stream data of it progress to the server every 5 second (can be modify) *ProgTrackClient.java

4 SOURCES

Discovery:

Michiel De Mey, Network Discovery using UDP Broadcast (Java) Retrieved from https://demey.io/network-discovery-using-udp-broadcast/

Transmission:

Mpetazzoni, Ttorrent, A Java Implementation of the BitTorrent Protocol Retrieved from https://github.com/mpetazzoni/ttorrent

Progress Tracking:

Jakob Jenkov, Multithreaded Server in Java Retrieved from

http://tutorials.jenkov.com/java-multithreaded-servers/multithreaded-server.html